A Cubical Approach to Synthetic Homotopy Theory

Guillaume Brunerie and Daniel R. Licata

Université de Nice and Wesleyan University

July 6, 2015
LICS 2015, Kyoto

Homotopy type theory

Homotopy theory is the study of spaces and continuous maps by way of their points, paths, paths between paths, and so on.

Homotopy type theory is an extension of Martin-Löf type theory by the *univalence axiom* and *higher inductive types* (HITs), which can be used to develop homotopy theory in type theory.

Type	$\leftrightarrow$	Space
Term of a type	$\leftrightarrow$	Point of a space
Identity type	$\leftrightarrow$	Type of paths
Equality proof	$\leftrightarrow$	Continuous path
Equality between equalities	$\leftrightarrow$	Homotopy between paths

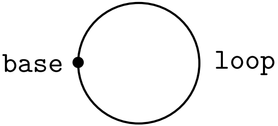
Univalence and HITs

- Martin-Löf's identity type is written `Path` with constructor `id`.
- Univalence says that the canonical map

$$\texttt{Path A B} \to \texttt{A} \simeq \texttt{B}$$

  has an inverse (for all types `A` and `B`)
- Higher inductive types are inductive types with constructors of paths

$$\begin{aligned} \texttt{S}^1 \ := \ & \texttt{base} : \texttt{S}^1 \\ & \texttt{loop} : \texttt{Path base base} \end{aligned}$$



Homotopy type theory

What we have currently:

- `Path` as an inductive family generated by `id`
- Univalence as an axiom
- HITs defined as axioms
- Definitional reduction rules for HITs for point-constructors
- Implemented and usable in Agda and Coq
- Consistent

But:

- No canonicity
- Missing reduction rules
- A lot of proofs contain much more bureaucracy than they should

Goal of the talk

Show a way to better handle the bureaucracy in what we have currently.

Motivating example: proving that the torus is equivalent to the product of two circles.

Application of a function to a path

We can apply a function `f : A → B` to a path `p : Path a a'`

```
ap f p : Path (f a) (f a')
ap f (id {a}) = id {f a}
```

Some properties:

```
ap-id p : Path (ap (λ x → x) p) p
ap-comp f g p : Path (ap (f ∘ g) p) (ap f (ap g p))
```

Dependent application

We can apply a dependent function

`f : (x : A) → B x`

to a path `p : Path a a'`

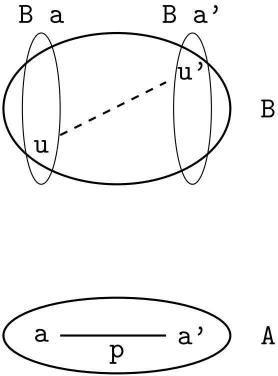
`apd f p : PathOver B p (f a) (f a')`

where the type

`PathOver B p u u'`

is the inductive family with constructor

`ido {u} : PathOver B id u u`



Paths in Sigma-types

For `(a, b)` and `(a', b')` two points in `Σ (x : A) (B x)`, the type

```
Path (a, b) (a', b')
```

is equivalent to the type

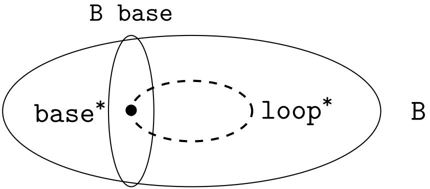
```
Σ (p : Path a a') (PathOver B p b b')
```

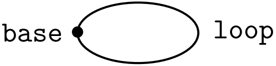
We write $(\mathtt{p},^1\ \mathtt{q})$ for the 1-dimensional pairing.

Circle

$$\mathtt{S}^1\texttt{-elim B base}^* \ \texttt{loop}^* \ : \ (\texttt{x} : \mathtt{S}^1) \to \texttt{B x}$$

$$(\texttt{base}^* : \texttt{B base} \text{ and } \texttt{loop}^* : \texttt{PathOver B loop base}^* \ \texttt{base}^*)$$





$$\mathtt{S}^1\texttt{-elim B base}^* \ \texttt{loop}^* \ \texttt{base} = \texttt{base}^*$$

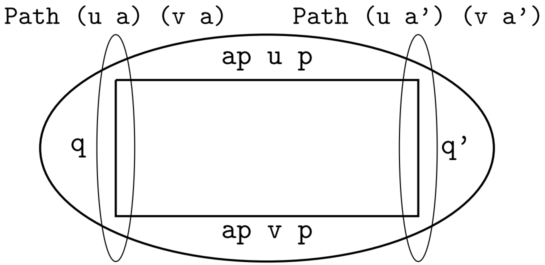
$$\beta\texttt{loop-elim} : \texttt{Path (apd (S}^1\texttt{-elim B base}^* \ \texttt{loop}^*\texttt{) loop) loop}^*$$

PathOver in Path

```
PathOver (λ x → Path (u x) (v x)) p q q'
```

is equivalent to the type of squares

```
Square q (ap u p) (ap v p) q'
```



We call `in-PathOver-Path` the map from `Square` to `PathOver`.

Squares

`Square` is an inductive family

```
Square : {A : Type} {w x y z : A}
         (l : Path w x) (t : Path w y)
         (b : Path x z) (r : Path y z)
         → Type
```

with constructor `ids : Square id id id id`.
We can define horizontal and vertical identities:

```
hid {p} : Square p id id p

vid {p} : Square id p p id
```

Dependent squares, cubes, etc.

We have 2-dimensional application $\mathtt{ap}^2$ and $\mathtt{apd}^2$, and 2-dimensional pairing $(\mathtt{p},^2\ \mathtt{q})$. We need for that dependent squares

```
SquareOver B sq l t b r
```

And we also need cubes

```
Cube left right back top bottom front
```

Cubes arise both from `PathOver` in a `Square` type and from `SquareOver` in a `Path` type.

Properties of 2-dimensional ap

Recall that

$$\texttt{ap-id p : Path (ap } (\lambda \texttt{ x} \to \texttt{x) p) p}$$

We cannot say $\texttt{ap}^2\texttt{-id sq : Path (ap}^2 \texttt{ } (\lambda \texttt{ x} \to \texttt{x) sq) sq}$ because the sides do not match.
We have to say

$$\begin{aligned}\texttt{ap}^2\texttt{-id sq : Cube (ap}^2 \texttt{ } (\lambda \texttt{ x} \to \texttt{x) sq) sq} \\ \texttt{(ap-id l) (ap-id t)} \\ \texttt{(ap-id b) (ap-id r)}\end{aligned}$$

The torus

The torus is the type `T` generated by

```
a : T
p : Path a a
q : Path a a
f : Square p q q p
```

We will use pattern matching notation to denote the use of eliminators.

From the torus to the product of circles

$$
\begin{aligned}
\texttt{t2c} &: \mathtt{T} \to \mathtt{S}^1 \times \mathtt{S}^1 \\
\texttt{t2c a} &= \texttt{(base, base)} \\
\texttt{ap t2c p} &= \texttt{(loop,}^1\ \texttt{id)} \\
\texttt{ap t2c q} &= \texttt{(id,}^1\ \texttt{loop)} \\
\texttt{ap}^2\ \texttt{t2c f} &= \texttt{(hid,}^2\ \texttt{vid)}
\end{aligned}
$$

The equations for `p` and `q` hold only up to a path and the equation for `f` holds up to a cube.

From the product of circles to the torus

```
        c2t : S¹ → S¹ → T
   c2t base = c2t-base
ap c2t loop = funext c2t-loop
```

```
        c2t-base : S¹ → T
   c2t-base base = a
ap c2t-base loop = q
```

```
      c2t-loop : (x : S¹) →
                 Path (c2t-base x) (c2t-base x)
 c2t-loop base = p
apd c2t-loop loop = in-PathOver-Path (βloop · f · βloop⁻¹)
```

Compositions

Using the same technique, we can easily construct:

```
c2t2c : (x y : S¹) → Path (t2c (c2t x y)) (x, y)
t2c2t : (z : T) → Path (c2t (t2c z)) z
```

Hence, $\mathtt{T}$ and $\mathtt{S}^1 \times \mathtt{S}^1$ are equivalent.

Conclusion

- Library functions inspired by cubical type theory make it easier to prove properties about 2-dimensional higher inductive types like the torus and about nested higher inductive types (the 3×3 lemma)
- A real cubical type theory, with the appropriate reduction rules and where dependent n-cubes are a primitive notion is still under investigation
- The code is available at `github.com/dlicata335/hott-agda` in the directory `lib/cubical/` and the file `homotopy/TS1S1.agda`